Socapel PAM
A Programmable Axes Manager

PAM CAN Bus Interface

User's Manual

Ordering Number: 006.8029.B
Rev. 4 June 1997

This upgraded and improved version replaces all the previous. We reserve the right to amend this document without prior notice and decline all responsibilities for eventual errors.

Atlas Copco Controls SA
En Montillier 4
CH-1303 PENTHAZ
Switzerland

Doc. No. 006.8029.B

© June 1997
by Atlas Copco Controls SA (previously SOCAPEL SA).
All rights reserved.

TABLE OF CONTENTS

1 INTRODUCTION ---------- 1-1

1.1 Manual Scope ---------- 1-1

1.2 Manual Organization and Contents ---------- 1-1

1.3 Definitions, Acronyms and Abbreviations ---------- 1-1

2 INTERFACE DESCRIPTION ---------- 2-1

2.1 CAN Summary Description ---------- 2-1

2.2 PAM CAN Bus Interface ---------- 2-1
2.2.1 DeviceNet Implementation Specification ---------- 2-2

2.3 Interface Board ---------- 2-2
2.3.1 User Connections ---------- 2-3
2.3.2 Interface Controller ---------- 2-4
2.3.3 Physical Interface ---------- 2-4
2.3.4 Node Address Selector ---------- 2-4
2.3.5 CAN Bus Interface Specifications ---------- 2-4

3 INTERFACE SOFTWARE ---------- 3-1

3.1 Introduction ---------- 3-1

3.2 Installation ---------- 3-1

3.3 Channel ---------- 3-1
3.3.1 Channel Declaration ---------- 3-2
3.3.2 Functions ---------- 3-5

3.4 Channel Variables ---------- 3-7
3.4.1 Channel Input Variable ---------- 3-7
3.4.2 Channel Input Set ---------- 3-8
3.4.3 Channel Output Variable ---------- 3-9
3.4.4 Channel Output Set ---------- 3-11
3.4.5 Function ---------- 3-13

3.5 Channel References ---------- 3-14
3.5.1 Multiple Linkages ---------- 3-17

3.6 Standard Declarations ---------- **3-19**
3.6.1 Standard Declaration for CAL with DBT Master ---------- 3-20
3.6.2 Standard Declaration for CAL without DBT Master ---------- 3-20
3.6.3 Standard Declaration for DeviceNet ---------- 3-20

3.7 Fatal Error Management ---------- **3-22**

4 EXAMPLE PROGRAMS ---------- **4-1**

4.1 For a CAL with DBT master ---------- **4-1**

4.2 For a CAL without DBT master ---------- **4-2**

4.3 For DeviceNet ---------- **4-6**

APPENDIX A :CAL SPECIFICATION (CIA/DS201..DS207) ---------- **4-1**

1 INTRODUCTION

1.1 MANUAL SCOPE

This manual is intended to serve as a reference for individuals who will be interfacing the PAM (Programmable Axis Manager) to a CAN (Controller Area Network) Bus utilizing the optional CAN Bus Interface for PAM. This manual provides details on the hardware and software interfacing aspects of the CAN Bus Interface for PAM, along with some example programs and applications information.

The scope of this manual is limited to the optional CAN Bus Interface. For a complete listing of all technical publications covering PAM and it's associated peripherals (i.e. ST1, Smart I-O, VME Bus Master and Simatic S5), refer to the Technical Publications Overview in the PAM User's Manual (document 006.8017.A).

1.2 MANUAL ORGANIZATION AND CONTENTS

This manual is organized into four chapters. The following is a summary of the manual contents by chapter:

Chapter 1	Description of manual organization and contents
Chapter 2	CAN Bus Interface description and specifications
Chapter 3	CAN Bus Interface software including declarations, parameters, functions and examples
Chapter 4	CAN Bus Interface, example program

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

CAL:

CAN Application Layer The application layer for CAN as specified by CiA.

CAN:

Controller Area Network. A network originally defined for use as a communication network for control applications in automobiles.

CMS:

CAN based Message Specification. One of the service elements of the application layer in the CAN Reference Model. CMS is a language that can describe how the functionality of a module can be accessed at its CAN interface.

COB:

Communication Object. A unit of transportation in a CAN Network. Data must be sent across a CAN Network inside a COB. There are 2032 different COB's in a CAN Network. A COB can contain at most 8 bytes of data.

COB-ID:

Each COB is uniquely identified in a CAN Network by a number called the COB Identifier (COB-ID). The COB-ID determines the priority of that COB for the MAC sub-layer.

REMOTE COB:

A COB whose transmission can be requested by another module.

DBT:

COB-ID Distributor. One of the service elements of the application layer in the CAN Reference Model. It's the responsibility of the DBT to distribute COB-ID's to COB's that are used by the CMS service element.

MAC:

Medium Access Control. One of the sub-layers of the Datalink Layer in the CAN Reference Model that controls who gets access to the medium to send a message.

NMT:

Network Management. One of the service elements of the application layer in the CAN Reference Model. The NMT serves to configure, initialize, and handle errors in a CAN network.

APPLICATION LAYER SERVICE ELEMENTS

The most important function of the application layer is to determine *what* an application can do with the communication environment The CAN application layer provides four application layer service elements

- CAN based Message Specification (CMS)
- Network Management (NMT)
- Distributor (DBT)
- (Layer Management)

CMS offers an open, object oriented environment to design user applications. CMS offers Variable-, Event-, and Domain- objects to design and specify how the functionality of a module can be accessed at its CAN interface. The Encoding Rules define how to encode and decode application data into the transfer syntax and vv.

NMT offers an open object oriented environment to let one module (the NMT Master) deal with the initialization and possible failures of the other modules (NMT Slaves).

The essential problem in defining an open CAN environment, is the *distribution of the COB identifiers.* A COB Identifier determines the priority for the MAC protocol of that COB. Therefore, the value of the identifiers may not be fixed by the suppliers of the different CAN modules since the systems integrator wants to have system-wide control over the priorities of the COB's. The DBT offers a *dynamic distribution of the identifiers* by one module (the DBT Master) to the other modules (DBT Slaves).

2 INTERFACE DESCRIPTION

2.1 CAN SUMMARY DESCRIPTION

The CAN concept was originally developed for use in the automotive industry to fulfill a need for a reliable, low cost serial data bus system for networking sensors, actuators and controllers within the automobile. Due to it's low cost, high real-time capabilities, excellent communications reliability and ability to function in harsh electrical environments, the CAN Bus is particularly well suited for networking intelligent I/O, sensors and actuators in industrial applications, and is becoming a communications standard in a number of segments within the industrial automation marketplace.

In a CAN network (see Figure 1), all peers (hosts and PAMs) receive all messages placed on the CAN Bus by any peer. Each peer then decides whether to discard or act upon an individual message based upon which peer originated the message and the nature of the message.

Each message placed on the CAN Bus includes three main parts:

- an identifier field which identifies the message originator and nature of the message,
- a data field which holds the data content of the message,
- a CRC (Cyclic Redundancy Check) field which contains a CRC code used to verify message integrity.

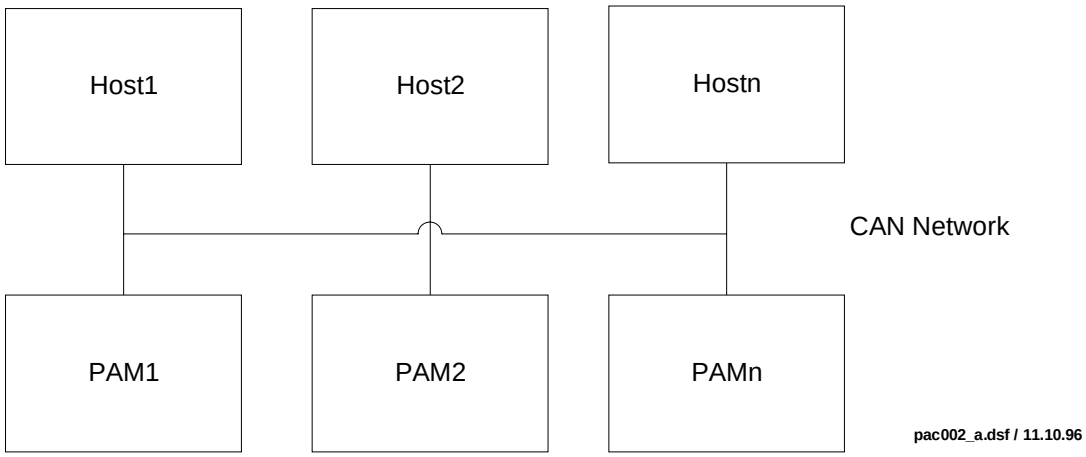


Figure 1 CAN Network

2.2 PAM CAN BUS INTERFACE

The PAM CAN Bus Interface utilizes standard CAN interface hardware and custom software which implements the CAL and DeviceNet communication protocols. An intermediate protocol handles transfers to and from the application via PAM Dualport memory.

Figure 2 illustrates the configuration. The PAM CAN interface hardware and software reside on a CAN Bus Interface board (see paragraph 2.3). On the PAM side, PAM firmware version 2.7 or higher supports the PAM CAN Bus interface.

The CAN Bus interface board is mounted in an available space in the PAM Stand-alone rack, above the power supply. Power and signal connections to PAM are made through the backpanel.

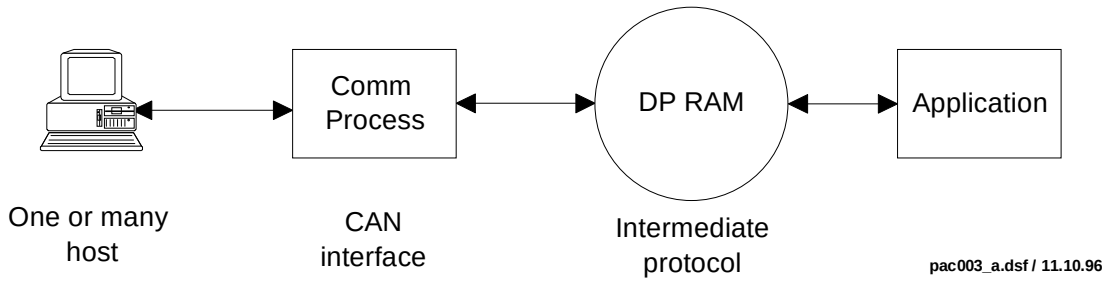


Figure 2 PAM CAN Bus Interface

2.2.1 DEVICENET IMPLEMENTATION SPECIFICATION

With DeviceNet, to improve the communication speed, the interface acts as Slave class2. This means only point-to-point communication with only one variable is available. Nevertheless, in that configuration, two addresses are automatically selected to have, in the applications point of view, two ports available. So, it is possible to communicate with two masters. In addition, an internal protocol allows transmission of more than one variable between one master and the PAM.

This protocol includes the COMM_ID of the variable (see §3.5 Channel References, page 3-14) in the CAN frame data field. To conserve 8 useful data bytes, fragmented frames are used.

(Start flag)	
CAN_Identifier	Low Byte
CAN_Identifier	High Byte
FragmentByte	First frame
COMM_ID	Low byte
COMM_ID	High byte
1. Databyte	Lowest byte
2. Databyte	
3. Databyte	
4. Databyte	
5. Databyte	
(CRC_Check)	
(StopFlag)	

First frame

(Start flag)	
CAN_Identifier	Low Byte
CAN_Identifier	High Byte
FragmentByte	Second and last frame
COMM_ID	Low byte
COMM_ID	High byte
6. Databyte	
7. Databyte	
8. Databyte	Highest byte
(CRC_Check)	
(StopFlag)	

Second frame

2.3 INTERFACE BOARD

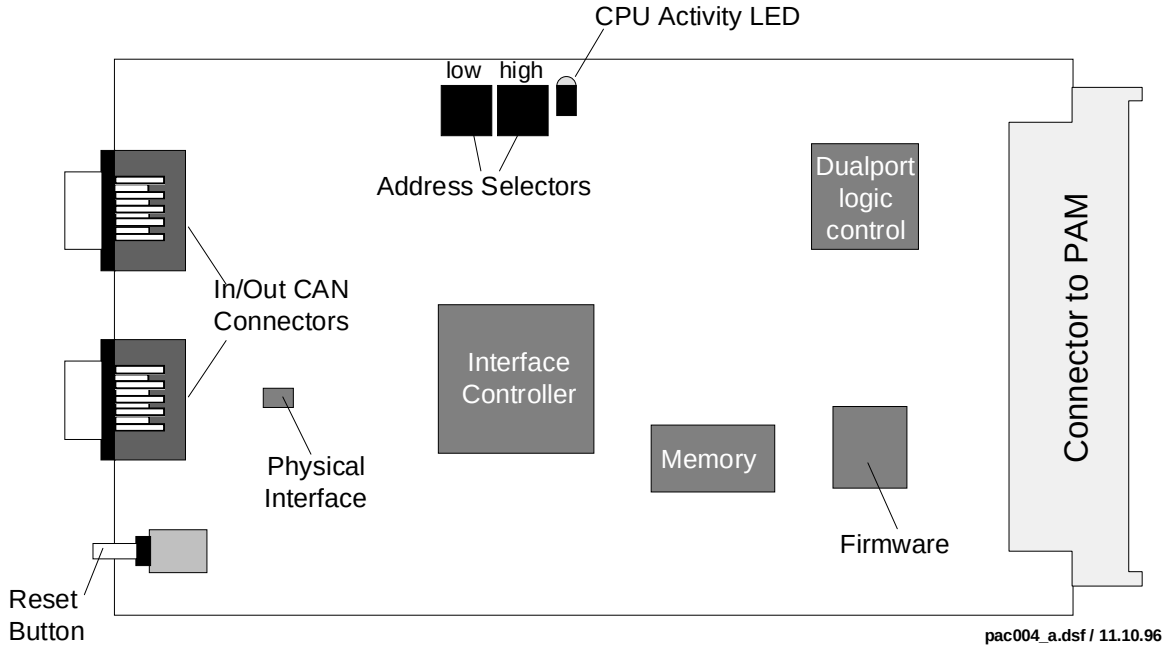


Figure 3 Interface board

2.3.1 USER CONNECTIONS

User connections are made via two nine pin D-SUB connectors (one male and one female) mounted on the front side of the board. One connector is for bus input, the other connector is for bus output. Equivalent pins on both connectors are jumpered together via the board. Connector pin assignments are listed in Table 1.

PIN	SIGNAL	REMARK
1	no connection	Reserved by CiA (CAN In Automation Users and Manufacturers Group)
2	CAN_L	CAN_L bus-line (dominant low)
3	GND	Ground
4	-	Reserved by CiA
5	-	Reserved by CiA
6	(GND)	Optional ground, internally connected to Pin 3
7	CAN_H	CAN_H bus-line (dominant high)
8	-	Reserved by CiA (error line)
9	V+CAN	Positive DC voltage (normally supplied to all nodes on the network by an external power supply)

Table 1 CAN Bus Interface Connector Pin Assignments

The CAN Bus connector is standardized by ISO Standard 11898.

Suitable mating connectors are available from a number of connector manufacturers, or they may be ordered from Atlas Copco Controls. Atlas Copco part numbers are listed in Table 2.

ATLAS COPCO ORDERING NUMBER	DESCRIPTION
505.3002	9 pin D-SUB connector, male
505.3003	9 pin D-SUB connector, female
505.3020	connector shell for circular cable (quantity = 2)

Table 2 Mating Connector Part Numbers

2.3.2 INTERFACE CONTROLLER

The CAN bus interface for PAM is based on the 80C592 microcontroller from Philips. It consists of a 80C51 microcontroller core and a 82C200 CAN controller integrated onto the same die. The 80C51 microcontroller utilizes an 8 bit architecture and is clocked at 16 MHz, providing a 750 nsec instruction cycle. The 82C200 CAN controller implements a Basic Standard CAN, which means the CAN identifier is limited to 11 bits.

2.3.3 PHYSICAL INTERFACE

The CAN Bus standard allows for variations in the implementation of the physical layer of the interface. The CAN Bus Interface board for PAM is supplied as standard with Philips PCA82C250T (PCA82C251T in version .B) and a 120 Ω terminal resistor.

2.3.4 NODE ADDRESS SELECTOR

Two coding wheels provide a way to assign a node address to the board. Each node on the CAN bus must have it's own address. The addresses are used during network initialization.

Address 0 is prohibited.

For DeviceNet, address 1 is also prohibited and two addresses are automatically selected.

Address on the coding wheel	Selected addresses
2	2 & 3
3	2 & 3
4	4 & 5
5	4 & 5
6	6 & 7
etc...	etc...

Table 3 DeviceNet selected addresses

2.3.5 CAN BUS INTERFACE SPECIFICATIONS

ABSOLUTE MINIMUM AND MAXIMUM VALUES

Any stress in excess of the MIN. or MAX. values may cause permanent damage.

Parameter	Min.	Max.	Unit
Storage temperature		80	°C
Operating temperature		40	°C
Supply voltage (VCC)		6.5	VDC
Voltage on the bus connections (V+CAN)	-0.3	35	VDC

NOMINAL VALUES

Parameter	Min.	Typ.	Max.	Unit
Current consumption		180	250	mA
Supply voltage (VCC)	4.75	5	5.25	VDC
Supply voltage (V+CAN)		24		
CAN controller clock frequency	-	16	-	MHz
Bus data rate	-	10, 20, 50, 100, 125, 250, 500, 1000 and others	-	kBits/s

3 INTERFACE SOFTWARE

3.1 INTRODUCTION

This chapter describes the process of initializing and interfacing to a CAN Bus from the perspective of the application program. Secondly, it provides definitions, syntax descriptions and examples of routines.

3.2 INSTALLATION

Depending on which configuration and protocol is used, different Boot- and Firmware are to be downloaded into PAM CAN Bus interface board.

	Bootware file	Firmware file
CAL with NMT	CBAZvvvv.CAN	CFAZvvvv.CAN
CAL without NMT	CBAZvvvv.CAN	CFBZvvvv.CAN
DeviceNet (slave class2)	CBAZvvvv.CAN	CFDZvvvv.CAN

vvvv is the version (e.g. 1000 means version 1.0_00).

Firmware files are located in ..\SOCATOOL\FIRMWARE and Bootware files in ..\SOCATOOL\BOOTWARE.

You must first download bootware and then firmware.

DOWNLOAD PROCEDURES

To download Bootware:

1. Click on to put PAM in BOOT state.
2. In the Advanced menu, select Download...
3. Set data type to CAN_BOOTWARE.
4. Browse to get the right CAN bootware file.
5. Click on OK.

To download Firmware:

1. Click on to put PAM in BOOT state.
2. In the Advanced menu, select Download...
3. Set data type to CAN_FIRMWARE.
4. Browse to get the right CAN firmware file.
5. Click on OK.

3.3 CHANNEL

3.3.1 CHANNEL DECLARATION

PURPOSE

The Channel Declaration establishes the operating parameters of the CAN Bus Interface during the initialization phase of application program execution. For most applications the default values of optional parameters should be used (see parameters description).

DECLARATION SYNTAX

CAN_CHANNEL <ChannelName> ;
[**SYNCHRO_TIMEOUT** = <Synchronization timeout value in ms>;]
[**WATCHDOG_TIMEOUT** = <Watchdog timeout value in ms>;]
[**GENERATE_VARID** = <**YES** | **NO**> ;]
[**MAX_PRIORITY_READ** = <Nb max priority successive reads> ;]
[**MAX_PRIORITY_FIFO_DATA_SIZE** = <Max data size in byte> ;]
[**MAX_FIFO_DATA_SIZE** = <Max data size in bytes> ;]
[**IN_PRIORITY_FIFO_SIZE** = <In priority FIFO size> ;
IN_FIFO_SIZE = <In FIFO size> ;
OUT_PRIORITY_FIFO_SIZE = <Out priority FIFO size> ;
OUT_FIFO_SIZE = <Out FIFO size> ;]
[**HIGH_PRIORITY_LEVEL** = <Highest priority level> ;
LOW_PRIORITY_LEVEL = <Lowest priority level> ;
LOW_LEVEL_PRIORITY_FIFO = <Lowest priority level for priority FIFO>;]
[**FATAL_ERROR_VARIABLE1** = <Fatal error channel variable name for msg 1> ;]
[**FATAL_ERROR_VARIABLE2** = <Fatal error channel variable name for msg 2> ;]
[**FATAL_ERROR_ROUTINE** = <Fatal error routine for message 1 and 2> ;]
[**NODE_NAME** = <Node name for first access> ;]
[**BAUDRATE** = <Baudrate for communication protocol> ;]
[**NODE_GUARDING_TIME** = <Node guarding time value in ms> ;
LIFE_TIME_FACTOR = <Life time factor> ;]
[**DEF_INHIBIT_TIME** = <Default Inhibit time in ms> ;]
[**DEF_PRIORITY** = <Default variable priority> ;]
[**DEF_FORMAT_NAME** = <**CAL_FIXED** | **CAL_FREE** > ;]
END_CHANNEL

PARAMETER DESCRIPTIONS

SYNCHRO_TIMEOUT (NA) [in milliseconds]

Timeout value used to wait for synchronization between PAM and the Can Bus interface through the dualport. The default value is 10000 (10 seconds).

WATCHDOG_TIMEOUT (NA) [in milliseconds]

Timeout value used to sense a breakdown in the communications process between the PAM and CAN Bus interface. The default value is 1000 (1second).

GENERATE_VARID (NA) [YES/NO]

This option must be set to **NO** if you have a dynamic identifier distributor and to **YES** if PAM must generate the communication identifiers. The default value is **NO**.

MAX_PRIORITY_READ (NA)

Specifies the maximum number of successive reads of the Priority Input FIFO allowed in one PAM cycle. When no value is specified, the default value is "no limit".

MAX_PRIORITY_FIFO_DATA_SIZE (NA) [in bytes, 4..8]

Specifies the size of data in the priority FIFOs. This size must be not smaller than 4. The default value is 8.

MAX_FIFO_DATA_SIZE (NA) [in bytes,4..8]

Specifies the size of data in the FIFOs. This size must be not smaller than 4. The default value is 8.

IN_PRIORITY_FIFO_SIZE (NA) [in blocks]

Specifies size of the input priority FIFO in number of block. This size must not be smaller than 10. The default value is 1/6 of dualport memory.

IN_FIFO_SIZE (NA) [in blocks]

Specifies size of the input FIFO in number of block. This size must not be smaller than 10. The default value is 1/3 of dualport memory.

OUT_PRIORITY_FIFO_SIZE (NA) [in blocks]

Specifies size of the output priority FIFO in number of block. This size must not be smaller than 10. the Default value is 1/6 of dualport memory.

OUT_FIFO_SIZE (NA) [in blocks]

Specifies size of the input FIFO in number of block. This size must not be smaller than 10. The default value is 1/3 of dualport memory.

HIGH_PRIORITY_LEVEL (NA) [0..7]

Specifies the highest priority level for the external communication protocol. The default value is 0.

LOW_PRIORITY_LEVEL (NA) [0..7]

Specifies the lowest priority level for the external communication protocol. The default value is 7.

LOW_LEVEL_PRIORITY_FIFO (NA) [LOW_PRIORITY_LEVEL..HIGH_PRIORITY_LEVEL]]

Specifies the lowest priority level for a variable to be put in the priority FIFO. The default value is **HIGH_PRIORITY_LEVEL**.

FATAL_ERROR_VARIABLE1 (NA)

The name of a Channel Variable (**CHANNEL_OUTPUT_VAR** or **CHANNEL_OUTPUT_SET**) to be transmitted when a fatal error occurs in PAM (see Fatal Error Management page 3-21).

FATAL_ERROR_VARIABLE2 (NA)

The name of a Channel Variable (**CHANNEL_OUTPUT_VAR** or **CHANNEL_OUTPUT_SET**) to be transmitted when a fatal error occurs in PAM (see Fatal Error Management page 3-21).

FATAL_ERROR_ROUTINE (NA)

The name of a **ROUTINE** to be executed when a fatal error occurs in PAM (see Fatal Error Management page 3-21).

NODE_NAME (NA)

This parameter specifies the PAM node name for connection by name. The default name is SocaPam.

BAUDRATE (NA) [in bits/s, 100000 or 200000 for CAL / 125000, 250000 or 500000 for DeviceNet]

Specifies the communication baudrate. The default value is 200000 (200 kbits/s).

NODE_GUARDING_TIMEOUT (NA) [in milliseconds]

For CAL only. This parameter give the node guarding time suggested to the NMT Master. Default value is none.

LIFE_TIME_FACTOR (NA)

For CAL only. This parameter gives the node life time factor suggested to the NMT Master. Default value is none.

DEF_INHIBIT_TIME (NA) [in milliseconds]

For CAL only. This parameter give the default inhibit time for Channel Output variables when **INHIBIT_TIME** is not specified in the Channel Output variable declaration.

Inhibit time specifies minimum delay between successive transmissions of the same variable. It provides a means to limit the communications channel bandwidth consumed by rapidly changing event driven channel output variables.

Default value is 0.

DEF_PRIORITY (NA) [0..7]

Specified the default Channel Variable priority when **PRIORITY** is not specified in the Channel Variable declaration.

Priority level specifies transmission priority requested for variable.

Default value is 7 (lowest).

DEF_FORMAT_NAME (NA) [CAL_FREE /CAL_FIXED]

For CAL only. This parameter give the default format name for **CHANNEL_REFERENCE** declarations when no **FORMAT_NAME** is specified. Format name specifies the format used for **CHANNEL_REFERENCE** names (see Channel References page 3-14).

Default value is **CAL_FREE**.

3.3.2 FUNCTIONS

The following inquire functions available for ***Channel***:

? ERROR

This Boolean inquire function is true if the channel is in error. The statement syntax is as follows:

<ChannelName> ? error

EXAMPLES:

```
IF CAN_MyChannel ? error THEN ...
EXCEPTION CAN_MyChannel ? error SEQUENCE SEQ_MyChanelErrorHandling ;
```

? ERROR_CODE (BOOLEAN)

This Boolean inquire function is true if the parameter error code matches the error state of the channel. The syntax of the statement is as follows:

<ChannelName> **? error_code**(<error code>)

EXAMPLES

```
IF CAN_MyChannel ? error_code(CHNL_NODE_INIT_ERROR)THEN ...
EXCEPTION CAN_MyChannel ? error_code(CHNL_NODE_INIT_ERROR) SEQUENCE
....
```

The file CHNLERR.SYS contains a listing of the error codes and their definitions. It is located in ..\SOCATOOL\APPINCS.

? ERROR_CODE (NUMERICAL)

This inquire function returns the current error code of the channel. The syntax of the statement is as follows:

<ChannelName> ? error_code

EXAMPLE

```
IF CAN_MyChannel ? error_code = CHNL_WATCHDOG_TIMEOUT THEN ...
```

RESET_ERROR_CODE (MASK)

This function reset the selected bit(s) of the current error code of the channel.

The syntax of the statement is as follows:

<ChannelName> <- reset_error_code (mask)

EXAMPLE

```
CAN_MyChannel <- reset_error_code(CHNL_WATCHDOG_TIMEOUT);
```

DISCONNECT_NODE (CAL Only)

This function stops the CAN communications.

The syntax of the statement is as follows:

<ChannelName> <- disconnect_node

EXAMPLE

```
CAN_MyChannel <- disconnect_node;
```

?NODE_ERROR_PORT (DeviceNet only)

This inquire function returns the current node guarding error port status of the interface. Return value is a status in which each bit represents a port (bit0 for port0, bit1 for port1)

The syntax of the statement is as follows:

<ChannelName> ? node_error_port

EXAMPLE

```
IF CAN_MyChannel ? node_error_port = 0x0001 THEN ...
```

RESET_NODE_ERROR_PORT (MASK) (DeviceNet only)

This function resets the selected bit(s) of the current node error port of the channel.

The syntax of the statement is as follows:

<ChannelName> <- reset_node_error_port (mask)

EXAMPLE

```
CAN_MyChannel <- reset_node_error_port(0x0001);
```

3.4 CHANNEL VARIABLES

Channel Variable declarations create a class of intermediate variables (Channel Variables) and define how data is mapped between Channel Variables and PAM Internal Variables (internal variables declared by the application program). **CHANNEL_REFERENCE** declarations (see paragraph 3.5) define the mapping of CAN Bus message data content to Channel Variables

In the Channel Variable descriptions, "input" means received by PAM and "output" means sent by PAM.

3.4.1 CHANNEL INPUT VARIABLE

The Channel Input Variable declaration creates a Channel Input Variable and maps it to a specified PAM Internal Variable.

DECLARATION SYNTAX

CHANNEL_INPUT_VAR <identifier> ;
[{**NUMBER** = <Number>|
NODES_GROUP = <Group identifier>} ;**]**
CHANNEL = <ChannelName> ;
[PRIORITY = <Priority level> ;**]**
VAR = <VariableName> ;
END

<Identifier>: Specifies the Channel Variable name.

<Number>= (NA), number (for i variables)

<Group identifier>= (NA) ,group identifier (for i variables)

*<ChannelName> : (NA), communications channel name from **CHANNEL** declaration.*

*<Priority level> : (NA), specifies priority level for the variable. Determines whether the variable is placed onto the Input Priority FIFO or the Input FIFO. Default Priority level is taken from **DEF_PRIORITY** in the **CHANNEL** declaration.*

<VariableName>: (NA), name of Internal PAM Variable into which Channel Input Variable maps.

EXAMPLE

This example maps Channel Variable "HIV_InFlag1" into the Internal Flag Variable "LFI_Flag1", and Channel Variable "HIV_Axis1Speed" into Internal Real Variable "LR1_Axis1Speed".

```
// Channel declaration
CAN_CHANNEL CAN_MyChannel ;
   GENERATE_VARID = NO ;
END_CHANNEL

// Linked application variables
INTERNAL FLAG_VAR LFI_Flag1 ;
INTERNAL REAL_VAR LRI_Axis1Speed ;
```

```
// Channel input variables
CHANNEL_INPUT_VAR HIV_InFlag1 ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 2 ;
   VAR = LFI_Flag1 ;
END
CHANNEL_INPUT_VAR HIV_Axis1Speed ;
   CHANNEL = CAN_MyChannel;
   VAR = LRI_Axis1Speed ;
END
// Channel references (see paragraph 3.5)
......
```

3.4.2 CHANNEL INPUT SET

The Channel Input Set declaration maps the content of a specified Channel Variable into multiple PAM Internal Variables.

DECLARATION SYNTAX

CHANNEL_INPUT_SET <identifier> ;
[{**NUMBER** = <Number>|
NODES_GROUP = <Group identifier>} ;**]**
CHANNEL = <ChannelName> ;
[PRIORITY = <Priority level> ;**]**
VARS = <VariableName**[**:<Bit range>**]**>,
{<VariableName**[**:<Bit range>**]**>} ;
END

<Identifier>: Specifies the Channel Variable name.

<Number>: (NA), number (for i variables)

<Group identifier>: (NA), group identifier (for i variables)

*<ChannelName> : (NA), communications channel name from **CHANNEL** declaration..*

*<Priority level> : (NA), specifies priority level for the variable. Determines whether the variable is placed onto the Input Priority FIFO or the Input FIFO. Default Priority level is taken from **DEF_PRIORITY** in the **CHANNEL** declaration.*

<VariableName> : (NA), name of Internal PAM Variables into which Channel Variable maps.

*<Bit range>: (NA), indicate the bits from the Channel Variable that map into this variable. For a **FLAG_VAR** only one bit, for a **WORD_VAR** a range of bit (Ex: 0..9).*

EXAMPLE:

In the following example, the **CHANNEL_INPUT_SET** declaration distributes the content of HIS_Msg among four internal variables as follows:

Variable name	value	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HIS_Msg		1	1	0	0	1	0	1	1	0	0	0	0	1	1	1	0
LFI_Flag1	TRUE	1															
LW1_SubCode1	14									0	0	0	0	1	1	1	0
LW1_SubCode2	75		1	0	0	1	0	1	1								
LW1_SubCode3	19214		1	0	0	1	0	1	1	0	0	0	0	1	1	1	0

The **CHANNEL_INPUT_SET** declaration provides great flexibility for re-arranging a message content into a format convenient for internal use.

```
// Channel declaration
   CAN_CHANNEL CAN_MyChannel ;
      GENERATE_VARID = NO ;
   END_CHANNEL

   //Linked application variables
   INTERNAL FLAG_VAR LFI_Flag1 ;
   INTERNAL WORD_VAR LWI_SubCode1 ;
   INTERNAL WORD_VAR LWI_SubCode2 ;
   INTERNAL WORD_VAR LWI_SubCode3 ;

   // Channel input variable set
   CHANNEL_INPUT_SET HIS_Msg ;
      CHANNEL = CAN_MyChannel;
      PRIORITY = 5 ;
      VARS = LFI_Flag1:15,
               LWI_SubCode1:0..7,
               LWI_SubCode2:8..14,
               LWI_SubCode3:0..14 ;
   END

   // Channel references (see paragraph 3.5)
```

3.4.3 CHANNEL OUTPUT VARIABLE

A Channel Output Variable declaration maps a single PAM Internal Variable into a specified Channel Output Variable.

DECLARATION SYNTAX

```
CHANNEL_OUTPUT_VAR <identifier> ;
   [{NUMBER = <Number>|
    NODES_GROUP = <Group identifier>} ;]
    CHANNEL = <ChannelName> ;
   [EVENT_DRIVEN = <YES | NO> ;]
   [PRIORITY = <Priority level> ;]
   [INHIBIT_TIME = <Inhibit time in ms> ;]
    VAR = <VariableName> ;
END
```

<Identifier>: Specifies the Channel Variable name.

<Number>= (NA), number (for i variables)

<Group identifier>= (NA), group identifier (for i variables)

*<ChannelName> : (NA), communications channel name from **CHANNEL** declaration.*

<VariableName> : (NA), internal PAM Variable which maps into Channel Variable.

***EVENT_DRIVEN** = <**YES**>: (NA), channel Variable is event-driven. PAM automatically updates and sends Channel Variable each time PAM Internal Variable value changes subject to constraint of **INHIBIT_TIME** parameter. **YES** is the default value.*

***EVENT_DRIVEN** = <**NO**>: (NA), application must update and send Channel Variable (see §3.4.5 Function p3-13).*

*<priority level>: (NA), specifies transmission priority requested for the variable. Default Priority level is taken from **DEF_PRIORITY** in the **CHANNEL** Declaration.*

*<inhibit time in ms>: (NA), For CAL only. Specifies minimum delay between successive transmissions of the same variable. This parameter provides a means to limit the communications channel bandwidth consumed by rapidly changing event driven channel output variables). Default inhibit time is taken from **DEF_INHIBIT_TIME** in the Channel declaration.*

EXAMPLE

In this example, Internal Flag Variable LFO_Flag1 is mapped to Channel Variable HOV_Flag1 and Internal Real Variable LRO_Axis1Speed is mapped to Channel Variable HOV_Axis1Speed.

```
// Channel declaration
CAN_CHANNEL CAN_MyChannel ;
END_CHANNEL

// Linked application variables
INTERNAL FLAG_VAR LFO_Flag1 ;
INTERNAL REAL_VAR LRO_Axis1Speed ;

// Channel output variable
CHANNEL_OUTPUT_VAR HOV_Flag1 ;
   CHANNEL = CAN_MyChannel;
   EVENT_DRIVEN = NO ;
   PRIORITY = 4 ;
   VAR = LFO_Flag1 ;
END

CHANNEL_OUTPUT_VAR HOV_Axis1Speed ;
   CHANNEL = CAN_MyChannel;
   VAR = LRO_Axis1Speed ;
END
```

3.4.4 CHANNEL OUTPUT SET

A Channel Output Set declaration maps multiple PAM Internal Variable into a specified Channel Variable. The **CHANNEL_OUTPUT_SET** is useful for constructing messages comprised of several pieces of data.

DECLARATION SYNTAX

CHANNEL_OUTPUT_SET <Identifier> ;
[{**NUMBER** = <Number>|
NODES_GROUP = <Group identifier>} ;**]**
{<VariableName**[**:<Bit range>**]**>} ;
[PRIORITY = <Priority level> ;**]**
[INHIBIT_TIME = <Inhibit time in ms> **;]**
CHANNEL = <ChannelName> ;
VARS = <VariableName**[**:<Bit range>**]**>,
END

<Identifier>: Specifies the channel variable name.

<Number> : (NA), number (for i variables)

<Group identifier>: (NA), group identifier (for i variables)

*<ChannelName> : (NA), communications channel name from **CHANNEL** declaration.*

<VariableName> : (NA), name of Internal PAM Variable which maps into Channel Variable.

*<Bit range>: (NA), indicate where the variable maps into the Channel Variable. For a **FLAG_VAR** only one bit, for a **WORD_VAR** a range of bit (E.g.: 0..9).*

The maximum size of a CHANNEL_OUTPUT_SET **is 64 bits.**

<Priority level>: *(NA), specifies transmission priority requested for the variable. Default Priority level is taken from* ***DEF_PRIORITY*** *in the* ***CHANNEL*** *declaration.*

<Inhibit time in ms>: *(NA), for CAL only. Specifies minimum delay between successive transmissions of the same variable. This parameter provides a means to limit the communications channel bandwidth consumed by rapidly sending channel output sets. Default inhibit time is taken from* ***DEF_INHIBIT_TIME*** *in the Channel declaration.*

As they are comprised of several variables, Channel Output Sets can not be event-driven.

EXAMPLE

In the following example, the **CHANNEL_OUTPUT _SET** declaration assembles the content of the Internal Word Variables LWO_Subcode1, LWO_Subcode2, LWO_Subcode3 and Internal Flag Variable LFO_Flag1 into **CHANNEL_OUTPUT_SET** HOS_Msg as illustrated in the following table:

Variable name	value	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LFO_Flag1	TRUE	1															
LWO_Subcode1	7									0	0	0	0	0	1	1	1
LWO_SubCode2	11		0	0	0	1	0	1	1								
LWO_SubCode3	24648		1	1	0	0	0	0	0	0	1	0	0	1	0	0	0
HOS_Msg		1	1	1	0	1	0	1	1	0	1	0	0	1	1	1	1

Note that the **CHANNEL_OUTPUT_SET** variable is the bitwise OR of all it's components.

```
// Channel declaration
CAN_CHANNEL CAN_MyChannel ;
   GENERATE_VARID = NO ;
END_CHANNEL

// Linked application variables
INTERNAL FLAG_VAR LFO_Flag1 ;
INTERNAL WORD_VAR LWO_SubCode1 ;
INTERNAL WORD_VAR LWO_SubCode2 ;
INTERNAL WORD_VAR LWO_SubCode3 ;
```

```
// Channel output variable set
CHANNEL_OUTPUT_SET HOS_Msg ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 5 ;
   VARS =   LFO_Flag1:15,
            LWO_SubCode1:0..7,
            LWO_SubCode2:8..14,
            LWO_SubCode3:0..14 ;
END

// How to use it in the application

LFO_Flag1 <- 1;         // First set the linked variables
LWO_Subcode1 <- 7;
LWO_Subcode2 <- 11;
LWO_Subcode3 <- 24648;

HOS_Msg <- send;        // Then send the channel variable to generate
                           the message(s)
```

3.4.5 FUNCTION

The following function is available for ***Channel Output Variables***:

SEND

This function is used to generate the messages to send a not event-driven output variable. The statement syntax is as follows:

<VariableName> <- send;

EXAMPLES:

```
// Once you have modified a linked application variable
LWO_MyWordVar <- 245;

// You can send the channel variable
HOV_MyOutWordVar <- send;
```

3.5 CHANNEL REFERENCES

Channel Reference declarations define the correlation between external messages and Channel Variables. In a CAN Bus Interface implementation, A **CHANNEL_REFERENCE** declaration cross-references the message identifiers of CAN Bus messages to corresponding Channel Variable(s) assigned to handle the message data content.

DECLARATION SYNTAX

CHANNEL_REFERENCE <Identifier> ;
    CHANNEL_VAR = <Channel var or set> ;
    [**FORMAT_NAME** = <Format name> ;]
    [**COMM_ID** = <communications identifier> ;]
END

<Identifier>: *Specifies the host variable name*
*In CAL system with a DBT master, it is used during initialization phase to define a COMM_ID. Then, in that case, it must be consistent with the **FORMAT_NAME** and with the name declared in the host. The two possible formats name are:*

- **CAL_FREE**: xxxxxxxxxxxx (12 characters). According to CAL specification (CiA/DS201..DS207), a "#" is added automatically in the interface board. If more than 12 characters are used, the name is truncated, if less than 12 character are used, "_" are added at the end.
- **CAL_FIXED**: pppxxxxxxxnnn, where, according to CAL specs:
  ppp (3 characters) must represent the profile,
  xxxxxxx (7 characters) must represent the variable name and
  nnn (3 digits) must represent the Device ID.

<Channel var or set>: *(NA), corresponding Channel Variable (or set) into which the host variable maps.*

<Format name>: *(NA), format name **CAL_FREE** or **CAL_FIXED** (CAL). The default format name is taken from **DEF_FORMAT_NAME** in the **CHANNEL** declaration.*

<communications identifier>: *(NA), host variable identifier assigned by application when communications identifiers are not assigned by DBT Bus Master.*

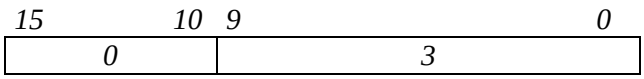
DeviceNet COMM_ID use

*In **DeviceNet**, the six most significant bits are used to select the port number to be used (then the master to communicate with). So available COMM_IDs are: 0x0000 to 0x03FE for port 0 and 0x0400 to 0x07FE for port1. 0x03FF and 0x7FF are reserved for COMM_ID of error response message (see Table 4)*

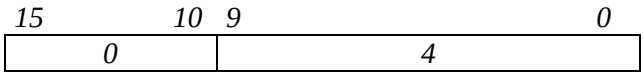
DeviceNet Master sends	*PAM CAN interface response message*	*PAM error*
Invalid fragmentation size	*0xAAAA*	*0BxxE01B* *Invalid fragmentation size*
Different COMM_ID in two fragmented frames	*0xBBBB*	*0BxxE01C* *Two different COMM_ID on port*
COMM_ID not defined as input	*0xCCCC*	*0BxxE01D* *Invalid input COMM_ID*
Invalid port number	*0xDDDD*	*0BxxE01E* *Invalid port number*
Correct Comm_ID	*Application data or 0x0000*	*none*
Default COMM_ID	*Application data or 0x0000*	*none*
Correct Comm_ID or default COMM_ID but input FIFO of PAM is full	*None*	*0BxxE014* *Input FIFO full*

TABLE 4 CAN BUS INTERFACE ERROR MESSAGE(COMM_ID =0x03FF or 0x 07FF depending on the port on which the error occurs)

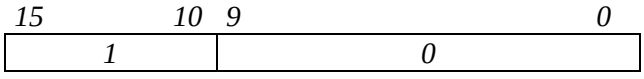
Example: How to declare 2 variables (#ID 3 and 4) for MasterA (via port0) and 2 variables(#ID 0 and 1) for MasterB (via port1).



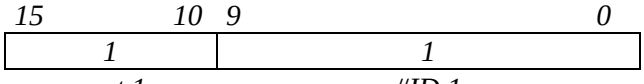
=> COMM_ID = 0x0003



=> COMM_ID = 0x0004



=> COMM_ID = 0x0400



=> COMM_ID = 0x0401

The maximum number of Channel_Reference is 400 for CAL and 500 for DeviceNet.

EXAMPLE

This example creates the following linkages between external, channel and application variables:

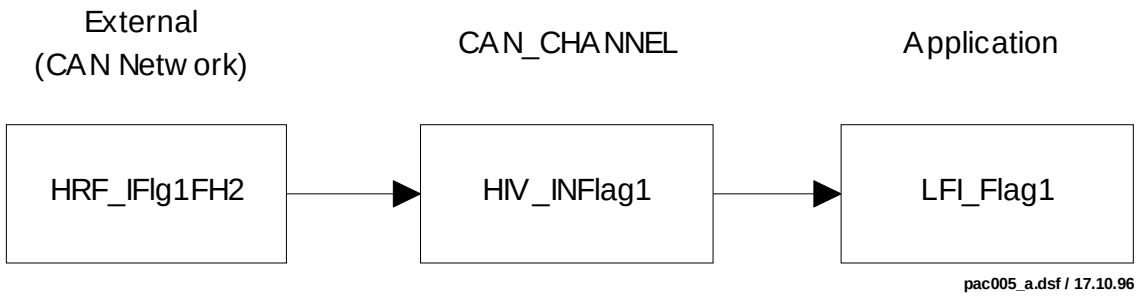


Figure 4 Linkage between Internal, Channel and External Communications Variables

```
// Channel declaration
   CAN_CHANNEL CAN_MyChannel ;
      GENERATE_VARID = NO ;
   END_CHANNEL

   //Linked application variables
   INTERNAL FLAG_VAR LFI_Flag1 ;

   // Channel input variable
   CHANNEL_INPUT_VAR HIV_InFlag1 ;
      CHANNEL = CAN_MyChannel;
      VAR = LFI_Flag1 ;
   END

   // Host reference to input variable
   CHANNEL_REFERENCE HRF_IFlg1FH2;//Input Flag1 From Host 2
      CHANNEL_VAR = HIV_InFlag1 ;
   END
```

3.5.1 MULTIPLE LINKAGES

The following examples illustrate the configuration when multiple Channel References are linked to a Channel Variable.

With CAL, broadcast is allowed. This means that you can declare the same variable in different Hosts. Note that every send of Channel Output Set HOS_Axis1Status produces only one HRF_OAx1STH message which can be read by both host1 and host2 (See Example 1).

With DeviceNet, broadcast is not allowed. Nevertheless, as two ports are available to communicate, you can send the same variable to two hosts linking two Channel References to one single Channel Variable (see Example 2).

EXAMPLE 1 (CAL)

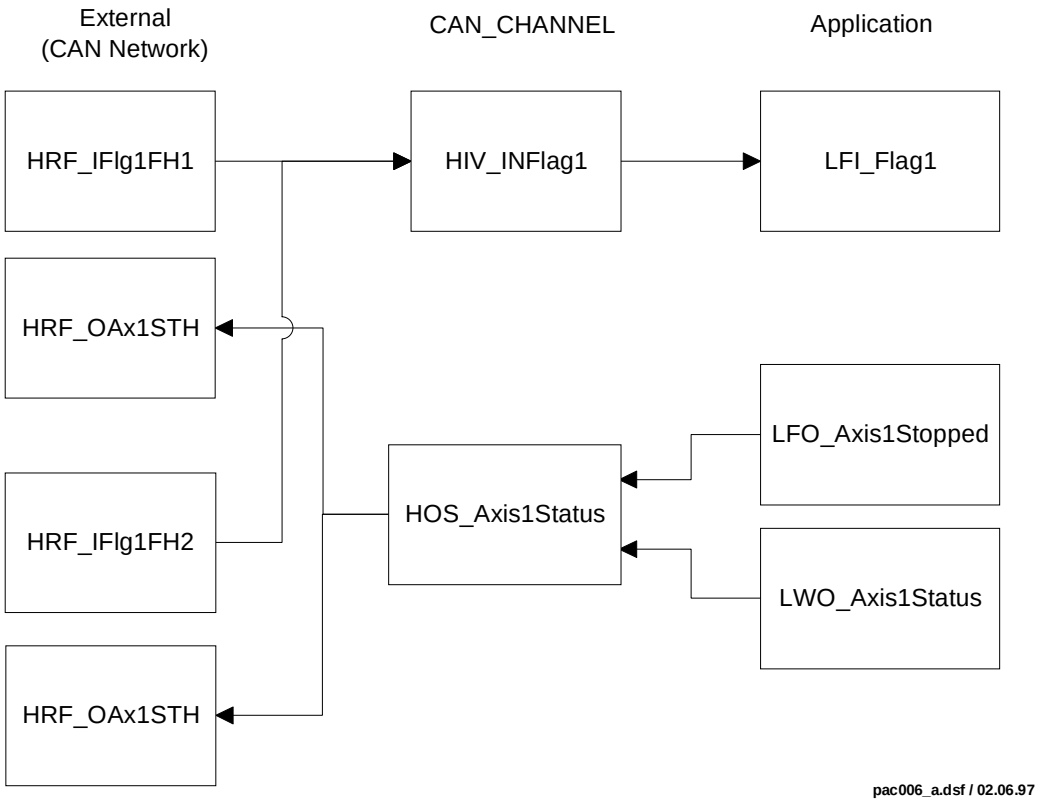


Figure 5 Multiple Linkages with CAL

```
// Channel declaration
CAN_CHANNEL CAN_MyChannel ;
   GENERATE_VARID = NO ;
END_CHANNEL

//Linked application variables
INTERNAL FLAG_VAR LFI_Flag1 ;
INTERNAL FLAG_VAR LFO_Axis1Stopped ;
INTERNAL WORD_VAR LWO_Axis1Status ;
```

```
// Channel input variable
CHANNEL_INPUT_VAR HIV_InFlag1 ;
   CHANNEL = CAN_MyChannel;
   VAR = LFI_Flag1 ;
END

// Host reference to input variable
CHANNEL_REFERENCE HRF_IFlg1FH2 ;// Input Flag1 From Host2
   CHANNEL_VAR = HIV_InFlag1 ;
END

CHANNEL_REFERENCE HRF_IFlg1FH3 ; // Input Flag1 From Host3
   CHANNEL_VAR = HIV_InFlag1 ;
END

// Channel output variable
CHANNEL_OUTPUT_SET HOS_Axis1Status ;
   CHANNEL = CAN_MyChannel;
   VARS =   LFO_Axis1Stopped :15,
            LWO_Axis1Status:0..14 ;
END

// Host reference to output variable
CHANNEL_REFERENCE HRF_OAx1STH ;//Output Axis1 Status To Hosts
   CHANNEL_VAR = HOS_Axis1Status ;
END
```

EXAMPLE 2 (DEVICENET)

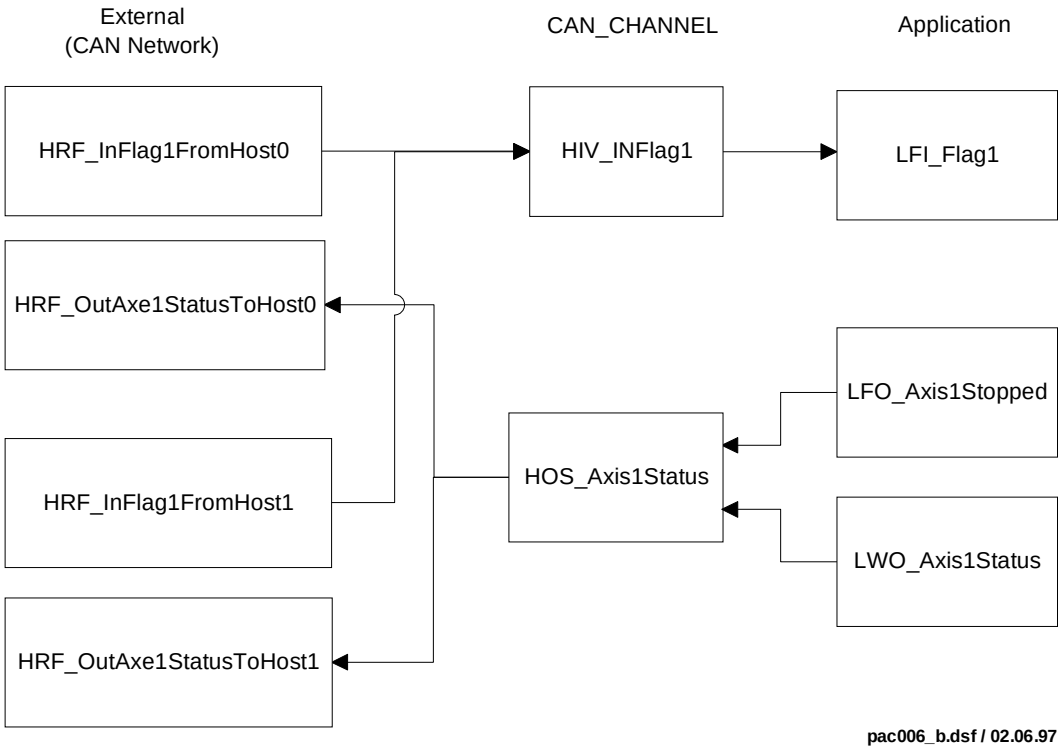


Figure 6 Multiple Linkages with DeviceNet

```
// Channel declaration
CAN_CHANNEL CAN_MyChannel ;
   BAUDRATE = 200000
   GENERATE_VARID = YES ;
END_CHANNEL

//Linked application variables
INTERNAL FLAG_VAR LFI_Flag1 ;
INTERNAL FLAG_VAR LFO_Axis1Stopped ;
INTERNAL WORD_VAR LWO_Axis1Status ;

// Channel input variable
CHANNEL_INPUT_VAR HIV_InFlag1 ;
   CHANNEL = CAN_MyChannel;
   VAR = LFI_Flag1 ;
END

// Host reference to input variable
CHANNEL_REFERENCE HRF_InFlag1FromHost0 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 0x0001 ;
END

CHANNEL_REFERENCE HRF_InFlag1FromHost1 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 0x0401 ;
END

// Channel output variable
CHANNEL_OUTPUT_SET HOS_Axis1Status ;
   CHANNEL = CAN_MyChannel;
   VARS =   LFO_Axis1Stopped :15,
            LWO_Axis1Status:0..14 ;
END

// Host reference to output variable
CHANNEL_REFERENCE HRF_OutAxe1StatusToHost0 ;
   CHANNEL_VAR = HOS_Axis1Status ;
   COMM_ID = 0x0002;
END

CHANNEL_REFERENCE HRF_OutAxe1StatusToHost1 ;
   CHANNEL_VAR = HOS_Axis1Status ;
   COMM_ID = 0x0402;
END
```

3.6 STANDARD DECLARATIONS

3.6.1 STANDARD DECLARATION FOR CAL WITH DBT MASTER

EXAMPLE

```
// Communication channel declaration
CAN_CHANNEL CAN_MyCALChannel ;
   BAUDRATE = 200000 ;
   GENERATE_VARID = NO ;
END_CHANNEL
//Linked application variables
INTERNAL WORD_VAR LWO_MyLinkedWordVariable ;
// channel output variables
CHANNEL_OUTPUT_VAR HOV_MyChannelOutputVariable ;
   CHANNEL = CAN_MyChannel ;
   VAR = LWO_MyLinkedWordVariable ;
END
// channel reference
CHANNEL_REFERENCE HRF_MyChaRef ;//12 characters according to CAL_FREE
   CHANNEL_VAR = HOV_MyChannelOutputVariable;
END
```

3.6.2 STANDARD DECLARATION FOR CAL WITHOUT DBT MASTER

EXAMPLE

```
// Communication channel declaration
CAN_CHANNEL CAN_MyCALChannel ;
   BAUDRATE = 200000 ;
   GENERATE_VARID = YES ;
END_CHANNEL
//Linked application variables
INTERNAL WORD_VAR LWO_MyLinkedWordOutputVariable ;
// channel output variables
CHANNEL_OUTPUT_VAR HOV_MyChannelOutputVariable ;
   CHANNEL = CAN_MyChannel ;
   VAR = LWO_MyLinkedWordOutputVariable ;
END
// channel reference (length is not important because without DBT
// Master, Only COMM_ID is used)
CHANNEL_REFERENCE HRF_MyChannelReference;
   CHANNEL_VAR = HOV_MyChannelOutputVariable ;
   COMM_ID = 4 ;
END
```

3.6.3 STANDARD DECLARATION FOR DEVICENET

EXAMPLE

```
// Communication channel declaration
CAN_CHANNEL CAN_DevNetChannel ;
   GENERATE_VARID = YES ;
   BAUDRATE = 125000 ;
END_CHANNEL

// application linked variables
INTERNAL WORD_VAR LWO_MyLinkedWordOutputVariable ;

// channel output variables
CHANNEL_OUTPUT_VAR HOV_MyChannelOutputVariable ;
   CHANNEL = CAN_MyChannel ;
   VAR = LWO_MyLinkedWordOutputVariable ;
END

// channel reference (length is not important because with DeviceNet
// only COMM_ID is used)
CHANNEL_REFERENCE HRF_MyChannelReference;
   CHANNEL_VAR = HOV_MyChannelOutputVariable ;
   COMM_ID = 4 ;
END
```

3.7 FATAL ERROR MANAGEMENT

When a fatal error occurs in PAM, it is possible, before the application is stopped, to send up to two messages. Size and reference of these messages are defined by the Channel Output Variable(s) (or Channel Output Set(s)) declared as:

- **FATAL_ERROR_VARIABLE1**
- **FATAL_ERROR_VARIABLE2**

in the **CHANNEL** declaration. These variables are constructed with four predefined 32 bits variables:

- **FatalError1High** and **FatalError1Low** for **FATAL_ERROR_VARIABLE1**
- **FatalError2High** and **FatalError2Low** for **FATAL_ERROR_VARIABLE2**

When a fatal error occurs, the contents of these predefined variable is automatically set:

- **FatalError1Low** <- PAMFatalErrorCode (Figure 7 shows the format used)
- **FatalError1High** <- Field1 (D1)
- **FatalError2Low** <- Field2 (D2)
- **FatalError2High** <- Field3 (D3)

The contents of these variables can also be set or modified in a routine which is declared in the **CHANNEL** declaration as **FATAL_ERROR_ROUTINE.** If it exists, this routine is automatically executed before the message are constructed (see Figure 8)

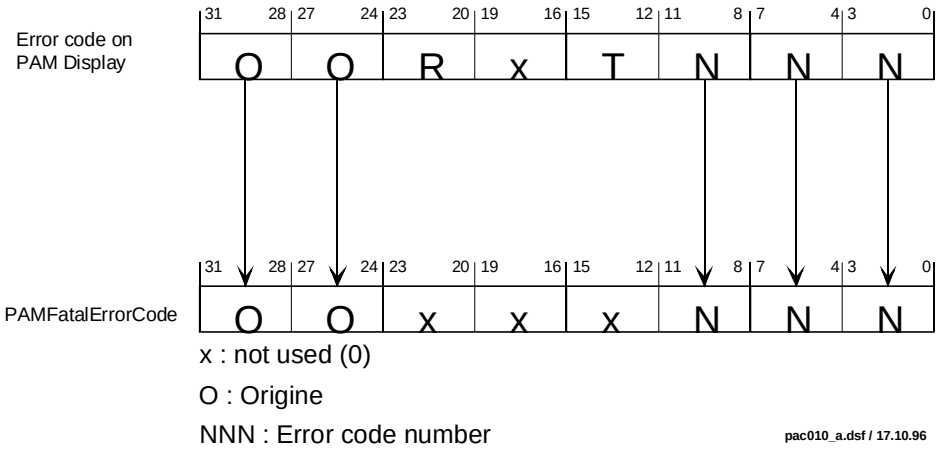

Error code on
PAM Display
31 28 27 24 23 20 19 16 15 12 11 8 7 4 3 0
O O R x T N N N
PAMFatalErrorCode
31 28 27 24 23 20 19 16 15 12 11 8 7 4 3 0
O O x x x N N N
x : not used (0)
O : Origine
NNN : Error code number
pac010_a.dsf / 17.10.96

Figure 7: PAMFatalErrorCode Format (see PAM V2.5 Reference Manual, appendix B for more details).

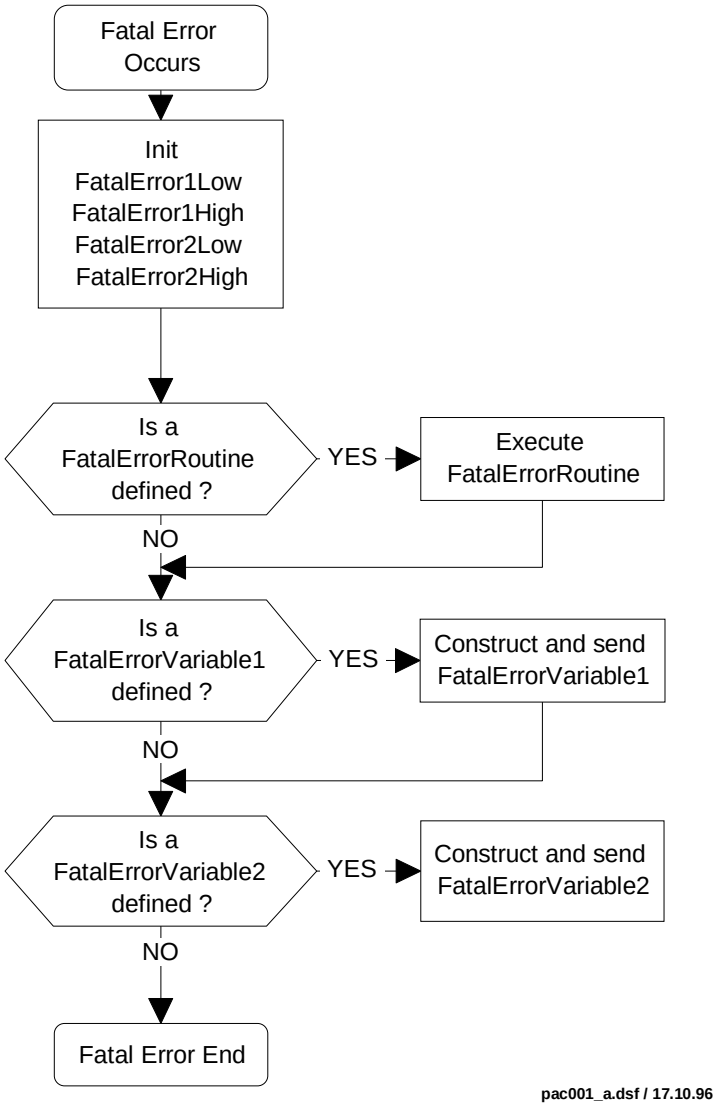


Figure 8 Fatal Error Management

EXAMPLES

The following example uses the default setting. It means that, when a fatal error occurs in PAM, the first message contains PAMFatalErrorCode and Field1 (D1). The second message contains Field2 (D2) and Field3 (D3).

```
// Communication channel declaration

CAN_CHANNEL CAN_MyChannel ;
   GENERATE_VARID = NO ;  // With dynamic distribution of identifier
   FATAL_ERROR_VARIABLE1 = HOS_FatalError ;// Both messages have same
   FATAL_ERROR_VARIABLE2 = HOS_FatalError ;// size and same reference
END_CHANNEL
```

```
// Application linked variables
INTERNAL WORD_VAR LWO_VarHigh ;
INTERNAL WORD_VAR LWO_VarLow ;

// Channel Output direct access message

CHANNEL_OUTPUT_SET HOS_FatalError ;
   CHANNEL = CAN_MyChannel;
   VARS = LWO_VarHigh:32..63,
          LWO_VarLow:0..31 ;
   END

// Host reference to Output direct access message

CHANNEL_REFERENCE HRF_FatalErr;
   CHANNEL_VAR = HOS_FatalError ;
END
```

In the following example, when a fatal error occurs in PAM, the first message is send to inform Host2 that PAM1 is in Fatal Error and a second one is send to give it the fatal error code.

```
// Communication channel declaration

CAN_CHANNEL CAN_MyChannel ;
   GENERATE_VARID = NO ;  // With dynamic distribution of identifier
   FATAL_ERROR_VARIABLE1 = HOS_PAMVariable ;// Both messages have same
   FATAL_ERROR_VARIABLE2 = HOS_PAMVariable ;// size and same reference
   FATAL_ERROR_ROUTINE = RTN_MyFatalErrorRoutine ;
END_CHANNEL

// Application linked variables
INTERNAL WORD_VAR LWO_VariableID ;
INTERNAL WORD_VAR LWO_VariableValue ;

// Channel Output direct access message

CHANNEL_OUTPUT_SET HOS_PAMVariable ;
   CHANNEL = CAN_MyChannel;
   VARS = LWO_VariableID:32..39,
          LWO_VariableValue:0..31 ;
   END

// Host reference to Output direct access message

CHANNEL_REFERENCE HRF_OPAMVTH2 ;//Output PAM Variable To Host2
   CHANNEL_VAR = HOS_PAMVariable ;
END

// Routines declaration

ROUTINES RGR_MyRoutines ;

   ROUTINE RTN_MyFatalErrorRoutine ;
      FatalError2High <- 0x13 ; // VariableID = PAM fatal error code
      FatalError2Low <- FatalError1Low ; //Pam Fatal Error code
      FatalError1High <- 0xAA ; // VariableID = PAMStatus
      FatalError1Low <- 0x002457FE ; // PAM in Fatal Error
   END_ROUTINES

END_ROUTINES

.......
```

4 EXAMPLE PROGRAMS

4.1 FOR A CAL WITH DBT MASTER

The declarations in this example program establish the communications linkages illustrated in Figure 9 during application execution.

```
// Communication channel declaration
CAN_CHANNEL CAN_MyChannel ;
   BAUDRATE = 200000 ;
   GENERATE_VARID = NO ;  // With dynamic distribution of identifier
END_CHANNEL

// Application variables
INTERNAL FLAG_VAR LFI_Flag1 ;
INTERNAL FLAG_VAR LFI_Axis1Start ;
INTERNAL FLAG_VAR LFI_Axis1Stop ;
INTERNAL WORD_VAR LWI_Axis1Cmd ;
INTERNAL REAL_VAR LRO_Axis1Speed ;
INTERNAL REAL_VAR LRO_Axis1Position ;
INTERNAL FLAG_VAR LFO_Axis1InError ;
INTERNAL WORD_VAR LWO_Axis1Status ;


// Channel input variable
CHANNEL_INPUT_VAR HIV_InFlag1 ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 7 ;
   VAR = LFI_Flag1 ;
END

// Host reference to input variable

CHANNEL_REFERENCE HRF_IFlg1FH1 ; // Input Flag1 From Host1
   CHANNEL_VAR = HIV_InFlag1 ;
END

CHANNEL_REFERENCE HRF_IFlg1FH2 ; // Input Flag1 From Host2
   CHANNEL_VAR = HIV_InFlag1 ;
END

CHANNEL_REFERENCE HRF_IFlg1FH3 ; // Input Flag1 From Host3
   CHANNEL_VAR = HIV_InFlag1 ;
END

// Channel input direct access message

CHANNEL_INPUT_SET HIS_Axis1 ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 4 ;
   VARS =  LFI_Axis1Start:0,
           LFI_Axis1Stop:1,
           LWI_Axis1Cmd; 0..31;
END
```

```
// Host reference to input direct access message

CHANNEL_REFERENCE HRF_IAx1MFH2 ; //Input Axis1 Message From Host2
   CHANNEL_VAR = HIS_Axis1 ;
END

CHANNEL_REFERENCE HRF_IAx1MFH3 ; //Input Axis1 Message From Host3
   CHANNEL_VAR = HIS_Axis1 ;
END

// Channel output variable

CHANNEL_OUTPUT_VAR HOV_Axis1Position ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 6 ;
   VAR = LRO_Axis1Position ;
END

CHANNEL_REFERENCE HRF_OAx1PTH ; //Output Axis1 Position To Hosts
   CHANNEL_VAR = HOV_Axis1Position ;
END

// Channel output variable not event-driven

CHANNEL_OUTPUT_VAR HOV_Axis1Speed ;
   CHANNEL = CAN_MyChannel;
   EVENT_DRIVEN = NO ;
   VAR = LRO_Axis1Speed ;
END

// Host reference to output variable

CHANNEL_REFERENCE HRF_OAx1STH ; //Output Axis1 Speed To Hosts
   CHANNEL_VAR = HOV_Axis1Speed ;
END

// Channel Output direct access message not event-driven

CHANNEL_OUTPUT_SET HOS_Axis1 ;
   CHANNEL = CAN_MyChannel;
   EVENT_DRIVEN = NO ;
   VARS =  LFO_Axis1InError:31,
           LWO_Axis1Status:0..30 ;
   END

// Host reference to Output direct access message

CHANNEL_REFERENCE HRF_OAx1MTH ; //Output Axis1 Message To Hosts
   CHANNEL_VAR = HOS_Axis1 ;
END

// Use in application

LRO_Axis1Position <- Axis1 ? position ;// Event if value changed
                                      => message send

LRO_Axis1Speed <- 120 ;// No event generated to DP -> no msg. sent
HOV_Axis1Speed <- send ;        // Manually send var. to DP

LFO_Axis1InError <- set ;
LWO_Axis1Status <- 12 ;
HOS_Axis1 <- send ;       // Manually send var. to DP
```

4.2 FOR A CAL WITHOUT DBT MASTER

The declarations in this example program establish the communications linkages illustrated in Figure 9 during application execution.

```
// Communication channel declaration
CAN_CHANNEL CAN_MyChannel ;
   BAUDRATE = 200000 ;
   GENERATE_VARID = YES ;  // Without dynamic distribution of identifier
END_CHANNEL

// Application variables
INTERNAL FLAG_VAR LFI_Flag1 ;
INTERNAL FLAG_VAR LFI_Axis1Start ;
INTERNAL FLAG_VAR LFI_Axis1Stop ;
INTERNAL WORD_VAR LWI_Axis1Cmd ;
INTERNAL REAL_VAR LRO_Axis1Speed ;
INTERNAL REAL_VAR LRO_Axis1Position ;
INTERNAL FLAG_VAR LFO_Axis1InError ;
INTERNAL WORD_VAR LWO_Axis1Status ;

// Channel input variable
CHANNEL_INPUT_VAR HIV_InFlag1 ;
   CHANNEL = CAN_MyChannel;
   VAR = LFI_Flag1 ;
END

// Host reference to input variable
CHANNEL_REFERENCE HRF_InFlag1FromHost1 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 1542 ;
END

CHANNEL_REFERENCE HRF_InFlag1FromHost2 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 1540 ;
END

CHANNEL_REFERENCE HRF_InFlag1FromHost3 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 1541 ;
END

// Channel input direct access message
CHANNEL_INPUT_SET HIS_Axis1 ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 4 ;
   VARS =  LFI_Axis1Start:0,
           LFI_Axis1Stop:1 ,
           LWI_Axis1Cmd; 0..31;
END

// Host reference to input direct access message
CHANNEL_REFERENCE HRF_InAx1MsgFromHost2 ;
   CHANNEL_VAR = HIS_Axis1 ;
   COMM_ID = 881 ;
END
```

```
CHANNEL_REFERENCE HRF_InAx1MsgFromHost3 ;
   CHANNEL_VAR = HIS_Axis1 ;
   COMM_ID = 882 ;
END

// Channel output variable
CHANNEL_OUTPUT_VAR HOV_Axis1Position ;
   CHANNEL = CAN_MyChannel;
   PRIORITY = 6 ;
   VAR = LRO_Axis1Position ;
END

CHANNEL_OUTPUT_VAR HOV_Axis1Speed ;
   CHANNEL = CAN_MyChannel;
   EVENT_DRIVEN = NO ;
   VAR = LRO_Axis1Speed ;
END

// Host reference to output variable
CHANNEL_REFERENCE HRF_OutAx1SpeedToHosts ;
   CHANNEL_VAR = HOV_Axis1Speed ;
   COMM_ID = 661 ;
END

CHANNEL_REFERENCE HRF_OutAx1PosToHosts;
   CHANNEL_VAR = HOV_Axis1Position ;
   COMM_ID = 1321 ;
END

// Channel Output direct access message
CHANNEL_OUTPUT_SET HOS_Axis1 ;
   CHANNEL = CAN_MyChannel;
   VARS =   LFO_Axis1InError:31,
            LWO_Axis1Status:0..30 ;
   END

// Host reference to Output direct access message
CHANNEL_REFERENCE HRF_OutAx1MsgToHosts ;
   CHANNEL_VAR = HOS_Axis1 ;
   COMM_ID = 1543 ;
END

// Use in application

LRO_Axis1Position <- 12.33 ;    // Event if value changed

LRO_Axis1Speed <- 12.33 ;       // No event generated to DP
HOV_Axis1Speed <- send ;        // Manually send var. to DP

LFO_Axis1InError <- set ;
LWO_Axis1Status <- 12 ;
HOS_Axis1 <- send ;             // Manually send var. to DP
```

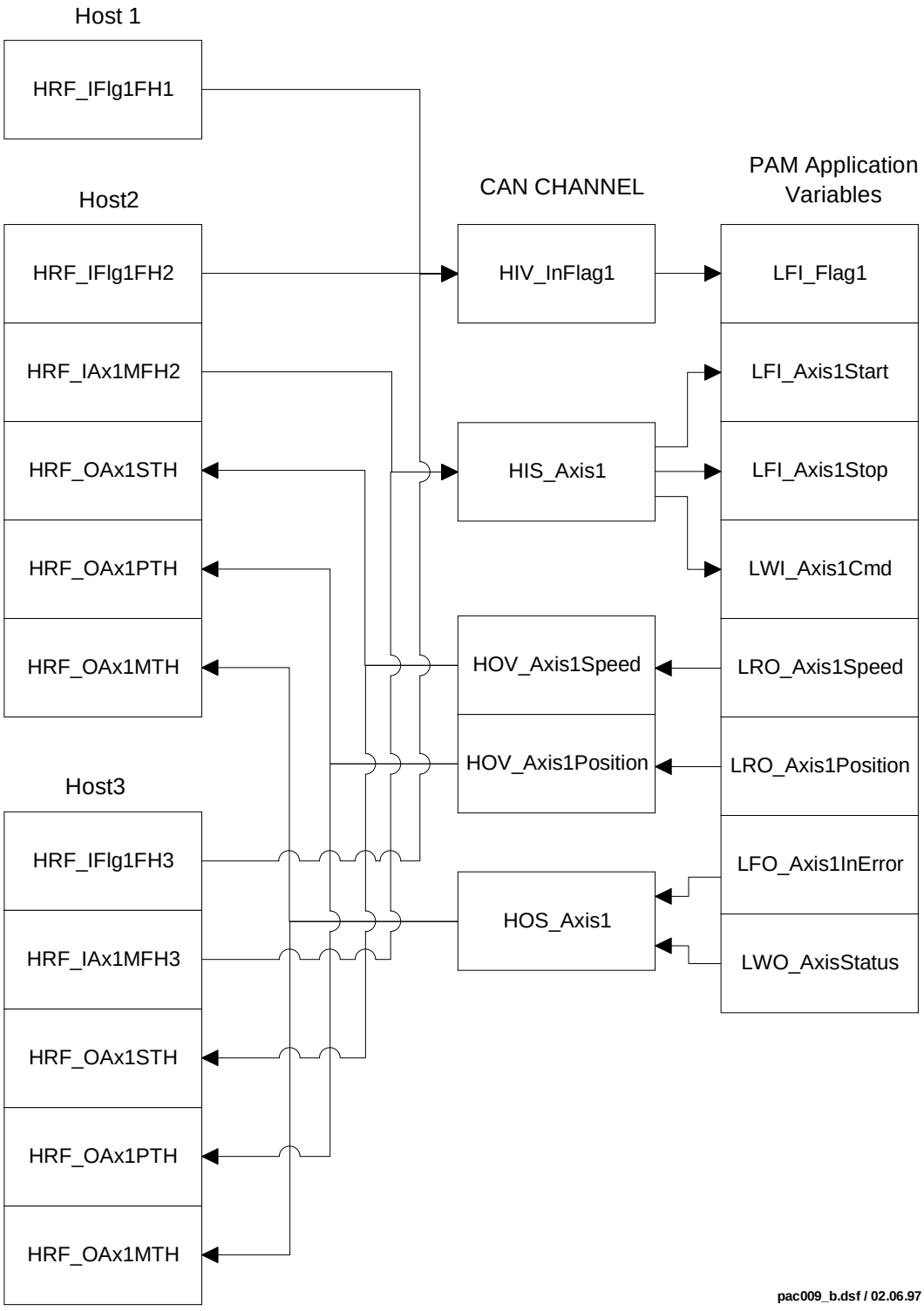


Figure 9 Linkages for Example Program (CAL)

4.3 FOR DEVICENET

The declarations in this example program establish the communications linkages illustrated in Figure 10 during application execution.

```
// Communication channel declaration
CAN_CHANNEL CAN_MyDevNetChannel ;
   BAUDRATE = 250000 ;
   GENERATE_VARID = YES ;  // Without dynamic distribution of identifier
END_CHANNEL

// Application variables
INTERNAL FLAG_VAR LFI_Flag1 ;
INTERNAL FLAG_VAR LFI_Axis1Start ;
INTERNAL FLAG_VAR LFI_Axis1Stop ;
INTERNAL FLAG_VAR LFO_Axis1InError ;
INTERNAL WORD_VAR LWI_Axis1Cmd ;
INTERNAL WORD_VAR LWO_Axis1Status ;
INTERNAL REAL_VAR LRO_Axis1Speed ;
INTERNAL REAL_VAR LRO_Axis1Position ;
// Channel input variable
CHANNEL_INPUT_VAR HIV_InFlag1 ;
   CHANNEL = CAN_MyDevNetChannel;
   VAR = LFI_Flag1 ;
END

// Host reference to input variable
CHANNEL_REFERENCE HRF_InFlag1FromHost0 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 0x01 ;
END

CHANNEL_REFERENCE HRF_InFlag1FromHost1 ;
   CHANNEL_VAR = HIV_InFlag1 ;
   COMM_ID = 0x401 ;
END

// Channel input direct access message
CHANNEL_INPUT_SET HIS_Axis1 ;
   CHANNEL = CAN_MyDevNetChannel;
   PRIORITY = 4 ;
   VARS =  LFI_Axis1Start:0,
           LFI_Axis1Stop:1 ,
           LWI_Axis1Cmd; 0..31;
END

// Host reference to input direct access message
CHANNEL_REFERENCE HRF_InAx1MsgFromHost0 ;
   CHANNEL_VAR = HIS_Axis1 ;
   COMM_ID = 0x02 ;
END

CHANNEL_REFERENCE HRF_InAx1MsgFromHost1 ;
   CHANNEL_VAR = HIS_Axis1 ;
   COMM_ID = 0x402 ;
END

// Channel output variable
CHANNEL_OUTPUT_VAR HOV_Axis1Position ;
   CHANNEL = CAN_MyDevNetChannel;
   PRIORITY = 6 ;
   VAR = LRO_Axis1Position ;
END

// Channel output variable not event-driven
CHANNEL_OUTPUT_VAR HOV_Axis1Speed ;
   CHANNEL = CAN_MyDevNetChannel;
```

```
   EVENT_DRIVEN = NO ;
   VAR = LRO_Axis1Speed ;
END

// Host references to output variable
CHANNEL_REFERENCE HRF_OutAx1PosToHost0 ;
   CHANNEL_VAR = HOV_Axis1Position ;
   COMM_ID = 0x0003 ;
END

CHANNEL_REFERENCE HRF_OAx1SpeTHost0 ;
   CHANNEL_VAR = HOV_Axis1Speed ;
   COMM_ID = 0x0004 ;
END

CHANNEL_REFERENCE HRF_OutAx1PosToHost1 ;
   CHANNEL_VAR = HOV_Axis1Position ;
   COMM_ID = 0x0403 ;
END

CHANNEL_REFERENCE HRF_OAx1SpeTHost1 ;
   CHANNEL_VAR = HOV_Axis1Speed ;
   COMM_ID = 0x0404 ;
END

// Channel Output direct access message
CHANNEL_OUTPUT_SET HOS_Axis1 ;
   CHANNEL = CAN_MyDevNetChannel;
   VARS =   LFO_Axis1InError:31,
            LWO_Axis1Status:0..30 ;
   END

// Host reference to Output direct access message
CHANNEL_REFERENCE HRF_OutAx1MsgToHost0 ;
   CHANNEL_VAR = HOS_Axis1 ;
   COMM_ID = 0x05 ;
END

CHANNEL_REFERENCE HRF_OutAx1MsgToHost1 ;
   CHANNEL_VAR = HOS_Axis1 ;
   COMM_ID = 0x405 ;
END
```

```
// Use in application
LRO_Axis1Position <- 12.33 ;          // Event if value changed
LRO_Axis1Speed <- 12.33 ;  // No event generated to DP
HOV_Axis1Speed <- send ;         // Manually send var. to DP
LFO_Axis1InError <- set ;
LWO_Axis1Status <- 12 ;
HOS_Axis1 <- send ;        // Manually send var. to DP
```

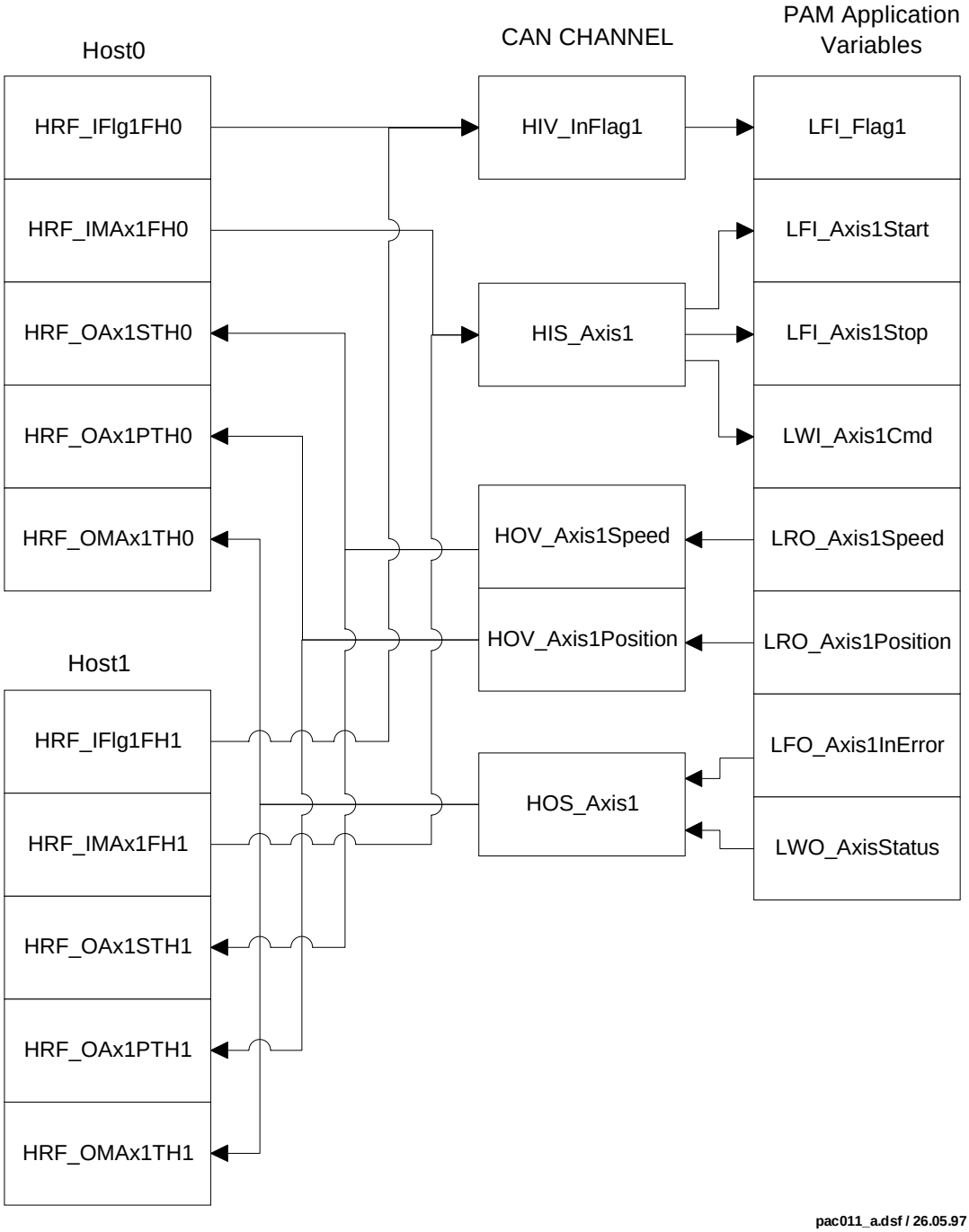


Figure 10 Linkages for Example Program (DeviceNet)

Appendix A :CAL SPECIFICATION (CIA/DS201..DS207)

1. The CAL master must be at least class 2 to have node guarding (CiA/DS203-1 3.4).
2. All variables are Write_only (CiA/DS202-2 4.3).
3. Dynamic identifier distribution is possible.